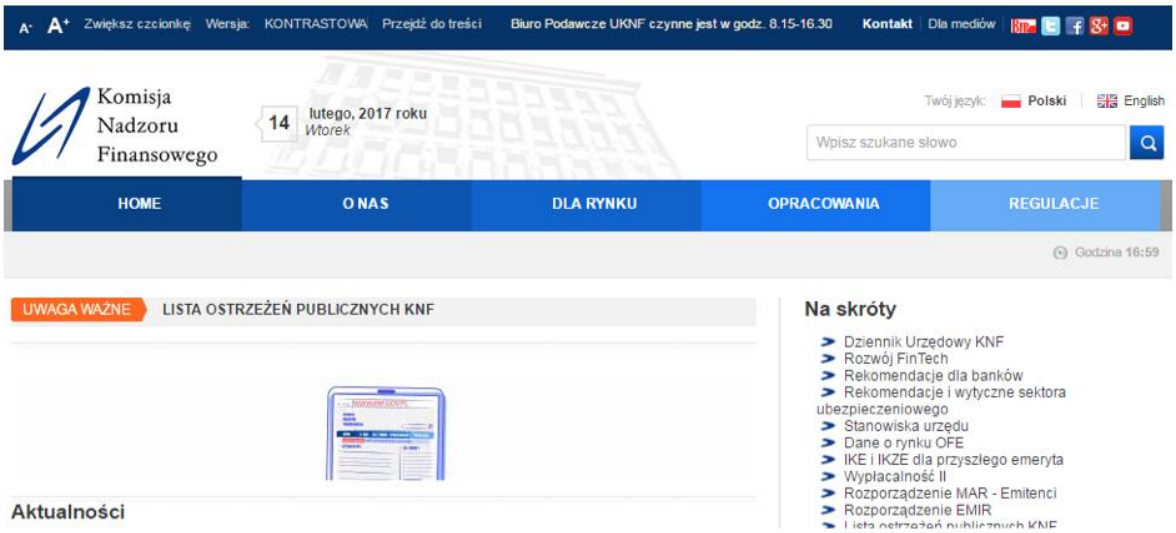
АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

Целевые атаки на польские банки: технический анализ

23 марта 2017 года

На польском портале ZaufanaTrzeciaStrona.pl, посвященном кибербезопасности, не так давно появилась новость об успешных атаках на польские банки (версия на английском здесь). Инцидент был охарактеризован как «серьезнейший». Данные подтвердили Symantec и BAE Systems. Список жертв пополнили учреждения из Мексики и Уругвая, а также другие цели по всему миру.



В этих атаках немало интересного – от целей и векторов заражения до особенностей вредоносных исполняемых файлов. Если первые два аспекта уже изучены, то вредоносный код не был детально исследован. В этом посте мы проведем технический анализ малвари.

Канал распространения

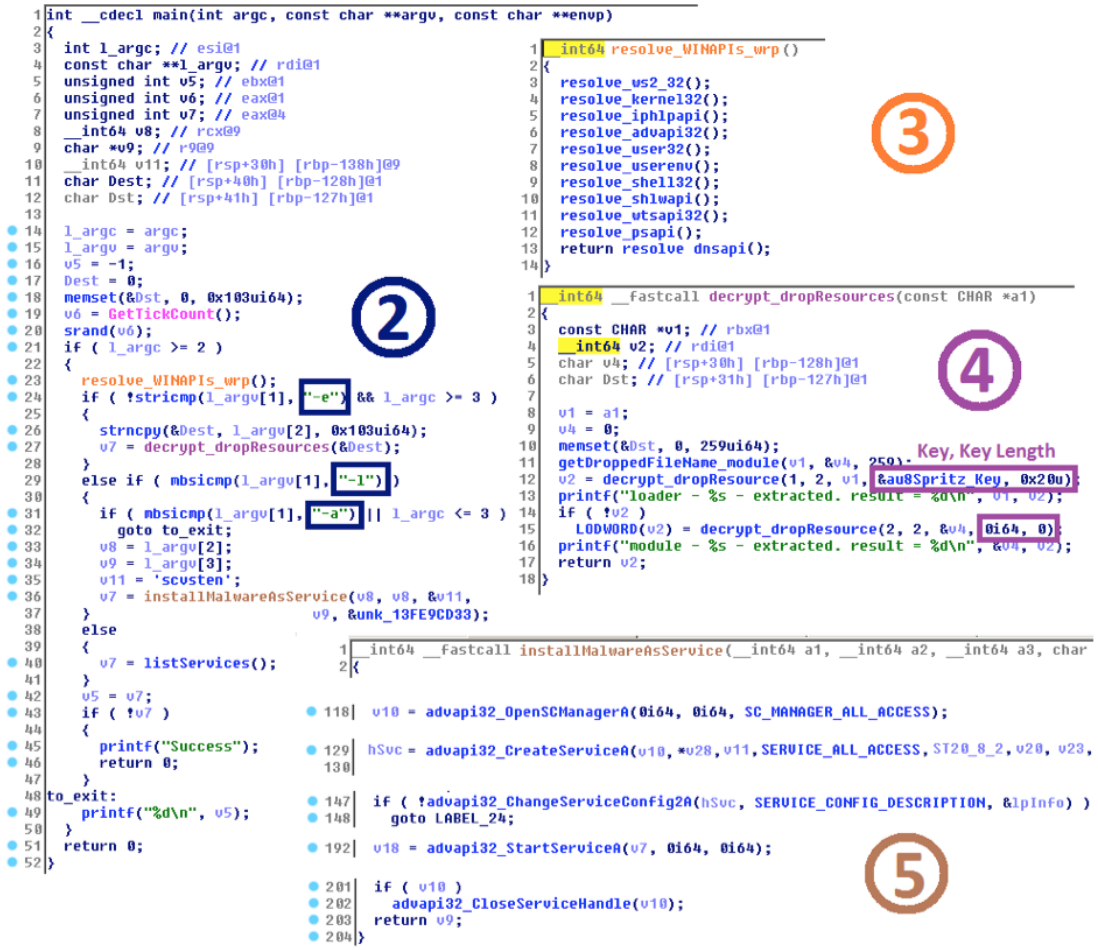
По данным ZaufanaTrzeciaStrona.pl, хакеры провели серию атак типа watering hole – заражение сайтов, которые посещают потенциальные жертвы. Скомпрометированный сайт перенаправляет пользователя на зараженную страницу с эксплойтом. В Польше отправной точкой выступал официальный сайт Комиссии по финансовому надзору (Komisji Nadzoru Finansowego).

По нашим данным, в Мексике с той же целью использовался сайт аналогичного ведомства (Национальной комиссии по банковской системе и ценным бумагам – Comisión Nacional Bancaria y de Valores). К сожалению, эта информация пока не подтверждена сервисами веб-трекинга или самим учреждением. Тем не менее, мы предполагаем, что перенаправление производилось с этого сайта:

Этап 1: Инъекция

Если эксплойт-кит успешно доставил троян, на компьютере жертвы исполняется вредоносная нагрузка – 64-битное консольное приложение. В отличие от инъекции, описанной BAE Systems, программа ожидает указания одного из трех аргументов: -l, -e или –a (раздел 2 на рисунке ниже).

В то время как параметр -l имеет то же значение, оставшиеся два необходимы для распаковки ресурсов следующего этапа (раздел 4) и для автозапуска одного из них как сервиса (раздел 5):



В разделе 5 инъекция пытается изменить конфигурацию системного сервиса, чтобы установить загрузчик как сервис. Конфигурация предназначена для автоматического запуска менеджером управления сервисами во время включения системы. Для этого нужны права администратора.

На первом этапе атаки угроза скрывает себя не так тщательно, как на последующих. Она даже содержит пространное заявление, предоставляющее информацию о статусе исполнения (в данном случае, об извлечении зашифрованных ресурсов; при этом информация об отладке, включая названия исходных функций, отсутствует).

Инъекция использует динамическую загрузку API вместо функций Windows в таблице импорта (подробнее об этом – в отчете компании Novetta Operation Blockbuster о группе хакеров Lazarus, страница 34). Раздел 3 на изображении выше демонстрирует оболочку этой функции, запускающей одну системную библиотеку за другой.

Похоже, что хакеры обозначают второй этап как «загрузчик», а третий, содержащий основной функционал вредоносного ПО – как «модуль». Загрузчик дешифруется, в то время как модуль извлекается и устанавливается без изменений. С целью маскировки файлы заимствуют время создания из системы shlwapi.dll.

Интересная особенность алгоритма шифрования – сравнительно новый потоковый шифр Spritz, напоминающий RC4. Реализация Spritz в языках программирования C и Python уже доступна, она соответствует следующему распакованному коду из инъекции:

```
void __fastcall decrypt(__int64 pOutput, unsigned int a2, __int64 key, unsigned __int8 Input)
{
  __int64 v4; // rbx@1
  unsigned int v5; // edi@1
  char State; // [rsp+20h] [rbp-138h]@1

  v4 = pOutput;
  v5 = a2;
  initstate_absorb_shuffle((__int64)&State, key, Input);
  squeeze((unsigned __int8 *)&State, v4, v5, v4);
}
```

```
__int64 __fastcall initstate_absorb_shuffle(  int64 State,   int64 a2,   int64 a3)
{
  unsigned __int8 *r; // rbx@1
  unsigned __int8 key_len; // r8@1
  __int64 key_array; // rcx@1
  unsigned __int8 *v6; // r9@1
  __int64 result; // rax@1

  r = State;
  initstate(State);
  result = absorb(key_array, v6, key_len)
  if ( r[0x104] )
    result = shuffle(r);
  return result;
}
```

```
__int64 __fastcall shuffle(unsigned __int8 *
{
  unsigned __int8 *v1; // rbx@1
  __int64 result; // rax@1

  v1 = State;
  whip();
  crush(v1);
  whip();
  crush(v1);
  result = whip();
  v1[0x104] = 0;
  return result;
}
```

Этап 2: Загрузчик

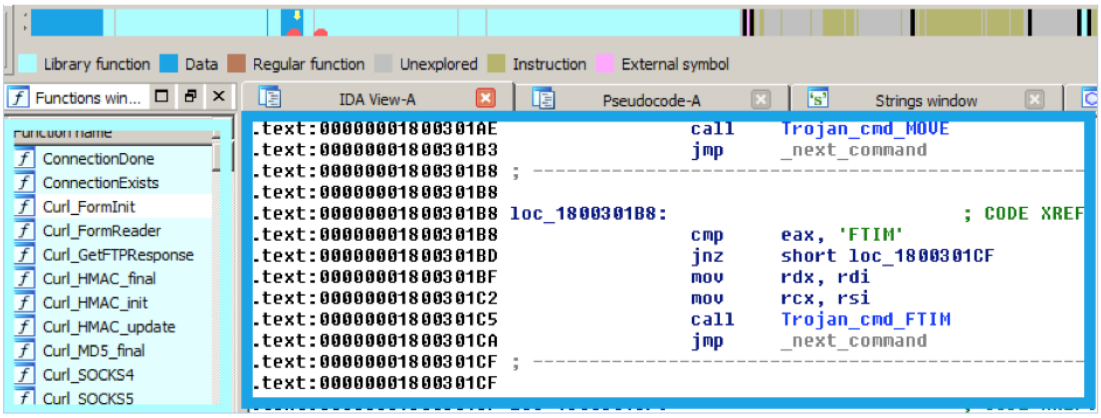
Угроза скрывает присутствие в зараженной системе. Загрузчик защищен платной утилитой Enigma Protector, модуль хранится зашифрованным. Как мы и ожидали, хакеры используют незарегистрированную копию 64-битной Enigma v1.31 – странно было бы рассчитывать, что квалифицированный автор допустит элементарную ошибку и поставит под угрозу раскрытия свою личность, используя официально зарегистрированную копию. (Напротив, использование взломанного или пиратского ПО, если оно доступно, не является необычным.) Хакеры, планирующие построить ботнет, как правило, не используют коммерческие архиваторы, поскольку часть антивирусных вендоров определяет их по вирусной сигнатуре. Следовательно, это ограничивает потенциальный размер ботнета. Но в случае целевой атаки в использовании такой защиты есть преимущества. Одно из них – восстановление исходного двоичного кода (каким он был до процесса маскировки) проще, чем когда-либо.

Создается впечатление, что угрозе подвержены только 64-битные версии Windows. Впечатление ошибочное, поскольку 32-битные модификации вредоносного ПО также обнаружены на компьютерах в пострадавших учреждениях. Несмотря на схожую структуру, 32-битная версия – не просто компиляция 64-битной, она немного отличается: стадии инъекции и загрузчика объединены в один этап, используется классическое шифрование RC4 вместо Spritz, модуль этого этапа хранится в системном реестре, а не в файловой системе. Версия используемого протектора Enigma – 3.7 с лицензией единой разработки, она явно используется для защиты двоичного кода 11 января 2017 года.

Этап 3: Модуль

Третий и завершающий этап – сравнительно большой модуль (около 730 Кб), который содержит основные функции вредоносной программы: взаимодействие с управляющим C&C сервером и получение команд операторов (хакеров). Модуль внедряет себя во все текущие сессии в скомпрометированной системе Windows.

В верхней строке представлены разные части бинарного кода: секции кода выделены голубым, секции данных – серо-желтым. Голубой цвет (в отличие от темно-голубого) – код, статистически связанный с существующими библиотеками. Помимо обычной среды выполнения C, мы обнаружили связь с кроссплатформенной библиотекой передачи файлов с открытым исходным кодом libcurl (версия 7.47.1, дата выпуска – 8 февраля 2016 года), а также с фрагментами кода проектов OpenSSL и XUnzip. Цветовой эффект в строке не генерируется автоматически, в этом случае нам нужно было явным образом отметить части, которые мы считаем связанными кодом библиотеки, и мы импортировали все имена функций. Темно-голубые разделы – код, написанный атакующими.



В модуле закодирован только один URL. Обмен данными шифруется. Мы не зафиксировали какое-либо взаимодействие, поскольку в период проведения анализа удаленный сервер не отвечал. Модуль поддерживает достаточно команд, чтобы можно классифицировать его как троян удаленного доступа. Набор команд следующий: “SLEP”, “HIBN”, “DRIV”, “DIR”, “DIRP”, “CHDR”, “RUN”, “RUNX”, “DEL”, “WIPE”, “MOVE”, “FTIM”, “NEWF”, “DOWN”, “ZDWN”, “UPLD”, “PVEW”, “PKIL”, “CMDL”, “DIE”, “GCFG”, “SCFG”, “TCON”, “PEEX”, “PEIN”. Назначение большинства команд очевидно: SLEP – перейти в режим ожидания, PKIL – уничтожить процесс, UPLD – скрытно выводить данные, DOWN – скачать, DEL – удалить файл и т.д. Не исключено, что исходные функции libcurl были адаптированы к задачам хакеров. Однако libcurl – это масштабный проект с сотнями участников, десятками тысяч строк кода и сотнями версий. Проверка и анализ осуществляется в настоящее время.

Инструментарий, напоминающий Lazarus

Исследователи из BAE Systems описывают 32-битную инъекцию, защищенную Enigma, следующим образом: «После распаковки ПО сбрасывает версию известной вредоносной программы, напоминающей часть инструментария группы хакеров Lazarus». Это подтверждают специалисты Symantec: «Некоторые строки кода имеют сходство с вредоносными программами кибергруппы Lazarus». Связь подтверждает и отчет компании Novetta, в частности, уже упомянутая динамическая загрузка API. На основе этих свидетельств мы охарактеризовали ключевые свойства вредоносного инструментария атакующих следующим образом:

1. Многоступенчатое вредоносное ПО, которое исполняется каскадом.
2. Начальный этап – консольное приложение, которое ожидает, как минимум, одного параметра.

3. WINAPI загружаются автоматически.
4. Потоковый шифр RC4 или другой подобный алгоритм с длинным ключом, который используется для дешифрования на следующем этапе.
5. Следующий этап(ы) – DLL библиотеки, которые загружаются как сервис с типом запуска SERVICE_AUTO_START (требуются права администратора).

Наши данные показывают недавнюю активность in-the-wild различных программ типа Lazarus. Тем не менее, чтобы представить четкую картину инцидента, нужно время для сбора дополнительной информации.

Странное открытие

В ходе исследования мы обнаружили интересный образец этого семейства вредоносного ПО. Это консольное приложение fdsvc.exe, ожидающее четырех параметров (сверяем со списком характеристик – 2), которое исполняется многоэтапно (1). Более того, приложение декодирует следующий этап при помощи RC4 с 32-битным ключом (4). Два свойства из списка выше (3 и 5) отсутствуют. С другой стороны, инструмент внедряет полезную нагрузку во все выполняющиеся сессии Windows. Полезная нагрузка статистически связана с libcurl v7.49.1.

Еще одна интересная особенность – команды хакера на финальном этапе. Используются команды «на русском» — написанные транслитом с кириллицы, но мало похожие на работу русскоговорящего автора.

```
do
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        pCmdLength_0 = 0;
        pCommand_0 = 0;
        memset(&Dst, 0, 0x3FFui64);
        if ( !Trojan_setsockopt_recv(l_s, &pCmdLength_0, 4, 60000) )
          return 0xFFFFFFFFi64;
        Trojan_decrypt(&pCmdLength_0, 4);
        l_pCmdLength_0 = pCmdLength_0;
        if ( !Trojan_setsockopt_recv(l_s, &pCommand_0, pCmdLength_0, 60000) )
          return 0xFFFFFFFFi64;
        Trojan_decrypt(&pCommand_0, l_pCmdLength_0);
        if ( stricmp(&pCommand_0, "ustanavlivat") )   ESTABLISH
          break;
        pCmdLength_1 = 0;
        pCommand_1 = 0;
        memset(&v11, 0, 0x103ui64);
        if ( !Trojan_setsockopt_recv(l_s, &pCmdLength_1, 4, 60000) )
          return 0xFFFFFFFFi64;
        Trojan_decrypt(&pCmdLength_1, 4);
        l_pCmdLength_1 = pCmdLength_1;
        if ( !Trojan_setsockopt_recv(l_s, &pCommand_1, pCmdLength_1, 60000) )
          return 0xFFFFFFFFi64;
        Trojan_decrypt(&pCommand_1, l_pCmdLength_1);
        memset(&::Dst, 0, 0x104ui64);
        strcpy_s(&::Dst, 0x104ui64, &pCommand_1);
      }
      if ( stricmp(&pCommand_0, "poluchit") )   GET
        break;
      pCmdLength_1 = strlen(&::Dst);
      if ( !Trojan_cmd_SEND(l_s, &pCmdLength_1, 4) || !Trojan_cmd_SEND(l_s, &::Dst, pCmdLength_1) )
        return 0xFFFFFFFFi64;
    }
    if ( stricmp(&pCommand_0, "pereslat") )   SEND
      break;
    pCmdLength_1 = 0;
    if ( !Trojan_setsockopt_recv(l_s, &pCmdLength_1, 4, 60000) )
      return 0xFFFFFFFFi64;
    Trojan_decrypt(&pCmdLength_1, 4);
    for ( i = 0; i < pCmdLength_1; ++i )
    {
      CreateThread(0i64, 0i64, start02_TCP_to_LINK, 0i64, 0, 0i64);
      Sleep(0x3E8u);
    }
  }                                   HOLD                    QUIT
}
while ( !stricmp(&pCommand_0, "derzhat") || stricmp(&pCommand_0, "vykhodit") )
v8 = *"vykhodit";
v9 = aVykhodit[8];
pCmdLength_1 = strlen(&v8);
if ( !Trojan_cmd_SEND(l_s, &pCmdLength_1, 4) || !Trojan_cmd_SEND(l_s, &v8, pCmdLength_1) )
  return 0xFFFFFFFFi64;
g_bExit = 1;
return 0i64;
}
```

```
signed __int64 __fastcall start02_TCP_to_LINK(
{
  signed __int64 v1; // rdi@1
  __int64 v3; // rax@3
  SOCKET v4; // rbx@3

  v1 = Trojan_Client2Connect("ssylka");
  if ( v1 == -1 )                          LINK
    return 0xFFFFFFFFi64;
  v3 = Trojan_socket_wrp_0();
  v4 = v3;
  if ( v3 == -1 )
    return 0xFFFFFFFFi64;
  Trojan_TCPMain(v1, v3);
  Trojan_Connect_Cleanup(v1);
  shutdown(v1, 2);
  closesocket(v1);
  shutdown(v4, 2);
  closesocket(v4);
  return 0i64;
}
```

Этот образец в очередной раз напоминает о том, как важно соблюдать осторожность с определением языковой принадлежности атакующих. Подобные «приманки» на ломаном русском вполне могут быть инсценировкой. Не вдаваясь в подробности филологического характера, можно вспомнить, что авторы вредоносного ПО чаще всего используют для обозначения команд числа или сокращения на латинице. Команда из 12 букв – это, как минимум, непрактично.

Заключение

Рассматривая образцы, рискнем предположить, что это не повторное использование кода, существовавшего задолго до атак на польские банки, и не забытый или остановленный проект. Более того, на протяжении многих недель мы наблюдали появление вредоносных программ, напоминающих «наши» образцы.

Таблица образцов и индикаторов

Образцы, использующиеся в атаках:

SHA1	Detection	Note
bedceafa2109139c793cb158cec9fa48f980ff2b	Win64/Spy.Banker.AX	Dropper;gpsvc.exe
aa115e6587a535146b7493d6c02896a7d322879e	Win64/Spy.Banker.AX	Enigma-protected loader
a107f1046f5224fdb3a5826fa6f940a981fe65a1	Win64/Spy.Banker.AX	Enigma-protected module; RAT; libcurl v. 7.47.1
4f0d7a33d23d53c0eb8b34d102cdd660fc5323a2	Win32/Spy.Banker.ADQH	32-bit Enigma-protected dropper;gpsvc.exe

Вредоносное ПО с транслитом в коде:

SHA1	Detection	Note
fa4f2e3f7c56210d1e380ec6d74a0b6dd776994b	Win64/Spy.Banker.AX	Dropper;fdsvc.exe
11568dffd6325ade217fbe49ce56a3ee5001cbcc	Win64/Spy.Banker.AX	Encrypted module;fdsvc.dll
e45ca027635f904101683413dd58fbd64d602ebe	Win64/Spy.Banker.AX	Decrypted module; RAT;libcurl v. 7.49.1 (*)
50b4f9a8fa6803f0aabb6fd9374244af40c2ba4c	Win32/Spy.Banker.ADRO	32-bit module; RAT;libcurl v. 7.49.1